Automated Abstractions for Patrolling Security Games

Nicola Basilico and **Nicola Gatti**
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133, Milano, Italy

Abstract

Recently, there has been a significant interest in studying *security games* to provide tools for addressing resource allocation problems in security applications. *Patrolling security games* (PSGs) constitute a special class of security games wherein the resources are mobile. One of the most relevant open problems in security games is the design of scalable algorithms to tackle realistic scenarios. While the literature mainly focuses on heuristics and decomposition techniques (e.g., double oracle), in this paper we provide, to the best of our knowledge, the first study on the use of *abstractions* in security games (specifically for PSGs) to design scalable algorithms. We define some classes of abstractions and we provide parametric algorithms to automatically generate abstractions. We show that abstractions allow one to relax the constraint of patrolling strategies' Markovianity (customary in PSGs) and to solve large game instances. We additionally pose the problem to search for the optimal abstraction and we develop an anytime algorithm to find it.

Introduction

The use of *algorithmic game theory* for security applications is receiving more and more attention in the scientific community. The key of its success resides in the modeling of security problems as interactive situations between a *defender* and an *attacker* wherein the attacker behaves optimally, not being committed to follow any predefined tactics as instead it is in non–game theoretic approaches. This allows one to capture realistic situations and find effective defensive strategies against rational attackers. The most known results provide randomized resource allocation techniques for problems wherein the defender uses *resources* to protect some *targets* with values from an attacker, and the resources are not sufficient to protect all the targets simultaneously, see, e.g., ARMOR (Pita et al. 2008) and RANGER (Tsai et al. 2010) systems.

Security games are customarily modeled as two–player (i.e., the defender and the attacker) games where the players act simultaneously, but one player (the attacker) can observe the strategy of the opponent (the defender) before acting, derive a correct belief over it, and act as best responder. This puts security games into the class of *leader–follower* games wherein the defender is the leader and the attacker is the follower. The appropriate solution concept is the *leader–follower* equilibrium, whose computation can be formulated as a multiple linear programming problem (Conitzer and Sandholm 2006). *Patrolling security games* (PSGs) constitute an interesting class of security games in which mobile resources are used to secure the targets (Basilico, Gatti, and Amigoni 2009). The peculiarity of PSGs resides in the game model. Differently from (Pita et al. 2008), PSGs present an extensive–form structure, the attacker being in the position to act during the execution of the defender's actions. The extensive–form structure introduces complications in the computation of the equilibrium that are commonly addressed by constraining the patrolling strategy to be Markovian. This allows one to formulate the resolution of a PSG as a multiple quadratic programming problem.

A crucial issue in the field of security games is the design of efficient algorithms capable to scale with realistic scenarios. In several practical situations the number of actions available to players are combinatorial and/or the attacker can be of different types having different preferences over the targets. In both cases, the number of mathematical programming problems to be solved is exponential and two approaches (i.e., *heuristics* and *decomposition*) are currently under investigation. Heuristic approaches are used to find approximate solutions (Paruchuri et al. 2008). Decomposition techniques, such as single and double oracle methods, are used to solve iteratively the game starting from a reduced game and incrementally augmenting it (Tsai et al. 2010). These techniques provide impressive results with simultaneous actions games, but are not effective with PSGs.

In this paper, we provide, to the best of our knowledge, the first study on *abstractions* for security games (specifically for PSGs). The idea behind abstractions is simple: given a game, a reduced *abstract* game is generated wherein some actions are somehow clustered, the abstract game is solved, and then the optimal agents' strategies in the abstract game are mapped onto the original one. Many works on abstractions have been carried out for extensive–form games, especially for poker (Gilpin and Sandholm 2007; Gilpin, Sandholm, and Sørensen 2008), but none for security games. We improve the state of the art as follows.

- We define different classes of abstractions for PSGs (i.e., *with* and *without* utility loss) and we provide automatic parametric tools to generate abstractions.

- We show that the exploitation of abstractions addresses the two main crucial issues of PSGs: they relax the constraint of employing only Markovian strategies (allowing algorithms to find better strategies) and allow algorithms to scale better (solving realistic scenarios).
- We pose the problem to search for the best abstraction (in terms of patroller's utility) and we provide an anytime algorithm to find it.

Patrolling Security Games

A PSG is a two–player game played on a graph–represented environment by a patroller and by an intruder. The game develops in turns at which both players act simultaneously. Formally, a PSG can be represented by a directed graph $G = (V, A, c, T, v_{\mathbf{p}}, v_{\mathbf{i}}, d)$, where V is the set of n vertices to be patrolled and A is the set of arcs connecting them. The function $c : A \rightarrow \mathbb{R}^+$ assigns each arc a weight. If not differently specified, $c(i, j) = 1\ \forall (i, j) \in A$. $T \subseteq V$ contains *targets*, i.e., vertices having some value for players and where intrusions can occur. The functions $v_{\mathbf{p}} : T \rightarrow \mathbb{R}^+$ and $v_{\mathbf{i}} : T \rightarrow \mathbb{R}^+$ assign each target a value for the patroller and for the intruder respectively; given a target t, $v_{\mathbf{p}}(t)$ and $v_{\mathbf{i}}(t)$ can be, in principle, different, see, e.g., (Paruchuri et al. 2007). The function $d : T \rightarrow \mathbb{N} \setminus \{0\}$ assigns each target t a *penetration time* $d(t)$, namely the number of turns that the intruder must spend to complete an intrusion in t. At each turn, the patroller can move from its current vertex i to an adjacent one (*move(i,j)* where $(i, j) \in A$) while the intruder can decide either to wait outside the environment (*wait*) or to attack a target t (*attack(t)*). (We will refer to arcs and patroller's actions without distinction.) There are three possible kinds of outcomes of this game:

- *intruder–capture*: the intruder plays action *attack(t)* at turn k and the patroller detects it in target t in the time interval $[k + 1, k + d(t)]$; agents' utilities are defined as $u_{\mathbf{p}} = \sum_{i \in T} v_{\mathbf{p}}(i)$ and $u_{\mathbf{i}} = -\epsilon$ for the patroller and intruder, respectively ($\epsilon > 0$ is a capture penalty);
- *penetration–t*: the intruder successfully completes the intrusion in target t; utilities are $u_{\mathbf{p}} = \sum_{i \in T \setminus \{t\}} v_{\mathbf{p}}(i)$ and $u_{\mathbf{i}} = v_{\mathbf{i}}(t)$;
- *no–attack*: for every turn k the intruder plays the *wait* action and never enters in the environment, in this case the game does not end in a finite number of turns; utilities are $u_{\mathbf{p}} = \sum_{i \in T} v_{\mathbf{p}}(i)$ and $u_{\mathbf{i}} = 0$.

In a PSG, the intruder is assumed to stay hidden outside the environment and observe the patroller's actions, deriving a correct belief over its strategy. Conversely, the patroller has no knowledge about the intruder's undertaken actions. This scenario leads to the adoption of a *leader–follower equilibrium* as the proper solution concept to be applied. As shown in (Basilico, Gatti, and Amigoni 2009) a PSG can be expressed in strategic form with the introduction of symmetries that force the patroller to repeat its strategy after a finite number of turns. More precisely, the *history* h_i of patroller's last $l = |h_i|$ visits ending in vertex i is introduced and the value of l is fixed. An action for the patroller is then the set of probabilities $\{\alpha(h_i, j)\}$ for all the possible histories of length l, where $\alpha(h_i, j)$ corresponds, in the original game representation, to the probability of playing *move(i,j)* every time the history of visits is h_i. An action for the intruder can be *stay–out* or *enter–when(i,t)*. With respect to the original game representation, these actions map to playing *wait* forever and playing *attack(t)* as soon as the patroller is in i, respectively. Once the reduction to strategic form is performed, the leader–follower equilibrium can be computed via quadratic programming. However, solving the quadratic optimization problem presents a large computational cost. This drawback introduces two main limitations on the problem instances that can be solved within a reasonable time: (a) only Markovian strategies are tractable ($l \leq 1$); (b) even with Markovian strategies, only small environments (i.e., with a small number of vertices and targets) can be addressed efficiently. Indeed, the size of the problem's instances grows exponentially with respect to l and n.

A technique that showed good results in dealing with the second limit is the removal of dominated actions, see (Basilico et al. 2009) for details. For the patroller this means to remove all those vertices that it will never visit. In our work, we will assume that V is the set of non–dominated vertices. For the intruder this means to discard all the *enter–when(i,t)* actions that are not candidate best responses. We denote with I the set of non dominated actions for the intruder. In what follows, we propose a technique based on game theoretical *abstractions* to address both limitations (a) and (b).

Abstraction Classes Definition

An abstraction algorithm is a technique that when applied to a game produces a smaller version of it. The seminal work on abstractions has been proposed in (Gilpin and Sandholm 2007), however since its application to PSG would not provide any reduction (PSG are general–sum games without chance nodes) we define abstractions specifically for PSGs. A PSG abstraction works on the patroller's action space as it is defined in the original representation of the game (i.e., the set of *move(i,j)* actions). The idea is to substitute subsets of actions with macro actions, with the assumption that playing a macro action means playing a fixed sequence of all the actions in the corresponding subset. In doing so, the vertices crossed during the execution of the macro actions are not present in the abstraction.

Example 1 *Consider Fig. 1. Ellipses are vertices of G (black nodes are targets). Vertex 6 appears in the original game, but not in the abstract game. Actions* move(4, 6) *and* move(6, 7) *are replaced by macro action* move–along(4,7) *and actions* move(7, 6) *and* move(6, 4) *are replaced by macro action* move–along(7,4).

Formally, a PSG abstraction can be represented with a set of vertices $X \subseteq V$ and a *procedure*, reported in Alg. 1, to update G given X, where X is the subset of vertices of V that are removed from G. We denote the abstract game as G^X and we refer to an abstraction by X.

When a PSG abstraction is applied, the number of vertices of the abstract game is (by definition) always smaller than in

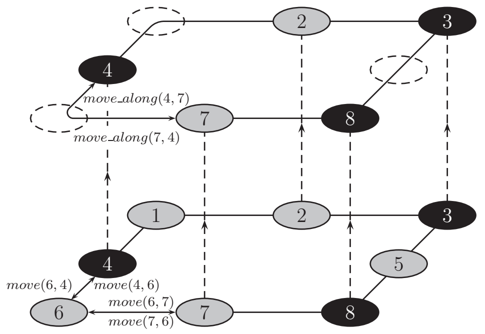

Figure 1: An example of PSG abstraction.

the original one. This is not true, in general, for the number of arcs (actions). However, to assure that an abstract game is easier to be solved than the original one we need that also the number of arcs decreases. We can do it by removing only vertices with indegree and outdegree equal to two. We define $\Lambda = \{x \in V \setminus T \mid deg^{+}(x) = deg^{-}(x) = 2\}$ as the set of candidates for X, assuming that $X \subseteq \Lambda$. The expected utilities of the patroller at the leader–follower equilibrium in different abstractions of the same game may be different. Thus, given an abstraction X, we denote as abstraction's *value* the patroller's expected utility and we denote it by $\nu(X)$. PSG abstractions can cope with both limitations (a) and (b).

Algorithm 1: ABSTRACTING(G, X)

```
1 $V \leftarrow V \setminus X$
2 for all $x$ in $X$ do
3     for all $(p, x) \in A$ do
4         for all $(x, s) \in A, s \neq p$ do
5             $A \leftarrow A \cup (p, s)$
6             $c(p, s) = c(p, x) + c(x, s)$
7         $A \leftarrow A \setminus (p, x)$
8     for all $(x, s) \in A$ do
9         $A \leftarrow A \setminus (x, s)$
```

On limitation (a): Markovian strategies in the abstract game cannot be generally mapped onto a Markovian strategy in the original game, but to strategies with a longer history.

Example 2 *Consider action* move–along(4,7) *in Fig. 1. Its actual realization cannot be obtained with a Markovian strategy in the original action space: in vertex* 6 *the patroller should consider also the previously visited vertex (*4 *or* 7*) to distinguish between* move–along(4,7) *and* move–along(7,4) *and, consequently, to decide where to move.*

Thus, computing a Markovian equilibrium strategy in the abstract game gives us, in general, a strategy in the original game that is not Markovian and that, therefore, may provide the patroller with larger utility.

On limitation (b): the problems to remove dominated strategies and compute an equilibrium can be formulated as in the non–abstract case, and the abstract game, having less vertices and arcs, leads to a smaller mathematical programming problem (in terms of variables and constraints) and then easier to be solved.

We focus on the strategic equivalence between a game G and its abstract version G^X for some abstraction X. To understand this we must consider how macro–actions affect the interaction between players. Macro–actions can provide the intruder with some advantage because playing fixed sequences of movements introduces predictability in the patroller's strategy. We capture this with the concept of *displacement* s_{it} of a vertex i for a target t.

Example 3 *Consider Fig. 1 where the patroller is in target* 4 *and plays action* move–along(4,7)*. As it moves in vertex* 6 *the intruder knows that the patroller moves toward* 7 *and not back to* 4*. Therefore, the intruder knows that vertex* 4 *will not be visited by the patroller for the next two turns, this is not the case in the original game.*

Given a target t and a vertex i we define the displacement s_{it} as the shortest distance between t and i computed on G^X. In general, if $s_{it} > dist(i,t)$ (where $dist(\cdot,\cdot)$ is the shortest distance on G) then the intruder could have a better revenue when playing *enter-when(i,t)* in G^X than in G. Given an abstraction X, displacements $\{s_{it}\}$ can be easily computed and their values define its abstraction class. There are three classes: *utility lossless*, *(potentially) utility loss*, and *inadmissible* abstractions. We describe them in increasing order in the number of potential vertices they can remove from G.

Utility lossless abstractions. The abstractions X belonging to this class are granted to have a value $\nu(X)$ non–smaller than the value $\nu(\varnothing)$ of the original game. We build an abstraction X of this class such that the set of intruder's non–dominated strategies is invariant w.r.t. the original game. The conditions X must satisfy can be expressed with a restriction on candidate abstract vertices Λ and with a set of linear constraints on displacements s_{it}. The set of candidate abstract vertices is restricted as follows:

$$\Lambda_{\text{LLA}} = \Lambda \cap \Big\{\{i \in V \setminus T \mid \forall t \in T,\ \textit{enter–when}(t,i) \notin I\} \cup \{t \in T \mid \forall w \in T,\ \textit{enter–when}(w,t) \notin I \text{ and } \forall i \in V \setminus T,\ \textit{enter–when}(t,i) \notin I\}\Big\}$$

That is, a vertex i is eligible for removal if, for every t', the intruder's action *enter–when (t', i)* is dominated; a target t is eligible for removal if, for every i', the intruder's action *enter–when (t, i')* is dominated. Then, given a vector $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$, abstraction $X = \{i \in V \mid x_i = 1\}$ is utility lossless if the following constraints are satisfied:

$$s_{it} = dist(i,t) \qquad \forall i \notin \Lambda_{\text{LLA}}, t \in T \tag{1}$$

$$s_{it} \geq dist(i,t) \qquad \forall i \in \Lambda_{\text{LLA}}, t \in T \tag{2}$$

$$s_{it} \leq dist(i,t) + nx_i \qquad \forall i \in \Lambda_{\text{LLA}}, t \in T \tag{3}$$

$$s_{it} \leq s_{jt} + 1 - n(1 - x_i) \qquad \forall i \in \Lambda_{\text{LLA}}, t \in T, j \in N(i,k), k \in D(i,t) \tag{4}$$

$$s_{it} \geq s_{jt} + 1 + n(1 - x_i) \qquad \forall i \in \Lambda_{\text{LLA}}, t \in T, j \in N(i,k), k \in D(i,t) \tag{5}$$

$$s_{it} \leq dist(k,t) \qquad \forall i \in \Lambda_{\text{LLA}}, t \in T, k \in D(i,t) \tag{6}$$

The set $D(i,t)$ denotes the vertices $j \in V$ such that *enter–when*$(t,j) \in I$ and this last action dominates *enter–when*(t,i), while $N(i,j)$ is the set of vertices adjacent to i in the shortest paths connecting i and j, such that if $k \in N(i,j)$ then $dist(k,j) < dist(i,j)$. Constraints (1) state that for every non–removable vertex the displacement for a target should be equal to the shortest distance between the two in G (this is the minimum displacement); constraints (2) allow displacement for removable vertices to be larger than the minimum value; constraints (3) assigns the minimum value to displacements of non removed vertices; constraints (4) and (5) define how to compute displacements given the set of removed vertices (notice that, since this formulation operates on G, it considers unitary costs on the arcs). Finally, constraints (6) set the upper bound for displacements of removable vertices. For a vertex i and target t, these values should not be greater than the distance between the target and the vertex j that dominates i in an attack to t.

We provide a proof sketch for utility lossless abstractions' properties. Constraints (6) avoid situations in which the intruder, exploiting the displacement of a removed vertex, finds a previously dominated action to be a possible best response in the abstract game. Therefore, the intruder's dominated actions I in G^X do not change with respect to G. Consider Fig. 1 and the set of probabilities $\{\alpha_{ij}\}$ as the leader–follower equilibrium of G. By assigning to macro–action *move–along(*$4,7$*)* the probability $\alpha_{4,6}$ and to macro-action *move-along(*$7,4$*)* the probability $\alpha_{7,6}$ we have that the probability for the patroller of reaching any target (i.e., the capture probability) from 4 or 7 when playing macro–actions is not smaller than when acting in G's action space. Vertex 6 is not considered since it is, like every other removed vertex, always dominated for the intruder (recall the definition of Λ_{LLA}). Therefore given a solution of G we can always find a solution of G^X which is not worse.

(Potentially) utility loss abstractions. The abstractions X belonging to this class are not granted to have a value $\nu(X)$ non–smaller than the value $\nu(\varnothing)$ of the original game. These abstractions guarantee only that, after their application, no target is *exposed* in G^X, namely that $\forall i \in V, t \in T$ $s_{it} \le d(t)$. If a target t is exposed then there is some i such that playing action *enter–when(t,i)* would result in a certain successful intrusion. The set of intruder's dominated actions is not an invariant. Differently from what the previous abstraction class does, this class does not pose any constraint on Λ, allowing for a larger number of removed vertices. Formally, the abstraction $X = \{i \in V \mid x_i = 1\}$ is lossy if the following linear constraints are satisfied:

constraints (1), (2), (3)

$$s_{it} \le s_{jt} + 1 - n(1 - x_i) \qquad \forall i \in \Lambda, t \in T, j \in N(i,t) \tag{7}$$

$$s_{it} \ge s_{,t} + 1 + n(1 - x_i) \qquad \forall i \in \Lambda, t \in T, j \in N(i,t) \tag{8}$$

$$s_{it} \le dist(i,t) + 1 - n(1 - x_i) \qquad \begin{array}{r}\forall i \in \Lambda, t \in T : N(i,t) = \varnothing,\\ \exists (i,k) \in A, dist(k,t) = dist(i,t)\end{array} \tag{9}$$

$$s_{it} \ge dist(i,t) + 1 + n(1 - x_i) \qquad \begin{array}{r}\forall i \in \Lambda, t \in T : N(i,t) = \varnothing,\\ \exists (i,k) \in A, dist(k,t) = dist(i,t)\end{array} \tag{10}$$

$$s_{it} = dist(i,t) \qquad \begin{array}{r}\forall i \in \Lambda, t \in T : N(i,t) = \varnothing,\\ \forall (i,k) \in A, dist(k,t) < dist(i,t)\end{array} \tag{11}$$

$$s_{it} \le d(t) \qquad \forall i \in \Lambda, t \in T \tag{12}$$

Constraints (7), (8), and (12) relax the corresponding constraints (1), (2), and (6) of lossless abstractions considering directly target t instead of the set $D(i,t)$. Constraints (9), (10), and (11) are analogous to (7) and (8), but they are applied when for a given vertex i and a target t there is not any adjacent vertex toward t (i.e., $N(i,t) = \varnothing$). This happens in the presence of cycles and precisely when i is the farthest vertex from t. Constraints (9) and (10) are applied when there exists a vertex k that is as far as i from t, while constraints (11) are applied when x is strictly the farthest. No guarantee can be provided over these abstractions' value. Notice that this abstractions class entirely contains the utility lossless abstractions class.

Inadmissible abstractions. An abstraction is inadmissible if for some i and t it holds that $s_{it} > d(t)$ (i.e., at least a target t is exposed). The inadmissible and utility loss classes are separate. Being these abstractions unsatisfactory, we will not consider this class in what follows.

Abstractions Generation and Evaluation

Given a game G, our interest is in choosing the abstraction with the maximum value among all those that are computable by a given deadline. However, the value and computational time can be known only *ex post* w.r.t. the resolution of the abstract game and it is not possible to evaluate all the abstractions, their number being exponential in the number of removable vertices (precisely, it is $O(2^{|\Lambda_{\text{LLA}}|})$ for lossless abstractions and $O(2^{|\Lambda|})$ for lossy abstractions). This problem is well known in the community studying abstractions (Gilpin, Sandholm, and Sørensen 2008). To provide a criterion to choose an abstraction we use two heuristics and a parameter $\lambda \in [0,1]$ that defines the tradeoff level between the two heuristics. The first heuristic aims at minimizing the computational time. The second heuristic aims at maximizing the value. With $\lambda = 0$ only the heuristic on value maximization is considered, while with $\lambda = 1$ *vice versa*. We provide furthermore a preliminary experimental evaluation to show how value and computational time vary with λ.

The heuristic for the computational time minimization is based on the number of vertices of an abstraction. The larger the number of removed vertices (it is reasonable to expect that) the smaller the computational time. We call *maximal* an abstraction that, among all the ones of its class, removes the maximum number of vertices. Maximal utility lossless and maximal lossy abstractions can be found with linear programs by maximizing $\sum_{i\in V} x_i$ under constraints (1), (2), (3), (7), (8) for the lossless and constraints (1), (2), (3), (5) for the lossy, respectively. Maximal abstractions provide the maximum potential saving in terms of computational time.

The definition of heuristics for the maximization of the value is more involved. The basic idea is to find, among all the abstractions with the same number of vertices, the one with the largest value. We capture this by using as heuristic the largest normalized displacement among those of all the vertices (the normalization is accomplished with the lower and upper bounds over the displacement of each vertex according to the used abstraction class). The intuition is that, since displacements introduce delays for the patroller in visiting targets, the larger the maximum normalized displacement the smaller the value. This heuristic can be formally captured by posing constraints over the displacements.

Consider the class of utility lossless abstractions. For a given λ we can find an abstraction as the result of the following linear program:

$$\max \sum_{i\in V} x_i \quad \text{s.t. constraints (1), (2), (3), (5)}$$

$$s_{it} \le (1-\lambda)dist(i,t) + \lambda dist(k,t) \qquad \forall i \in \Lambda_{LLA}, t \in T, k \in D(i,t) \tag{13}$$

Constraints (13) generalize constraints (5) by making the upper bounds on displacements depending on λ. In particular with $\lambda = 0$ displacements are forced to assume the minimum value and, consequently, the resolution of the above program returns the original game. With $\lambda = 1$ displacements can assume the maximum value allowed in a lossless abstraction and, maximizing the objective function, the resolution returns the maximal lossless abstraction (indeed, in this case constraints (13) coincide with constraints (5)). For increasing values of λ the resulting abstraction tends to remove more vertices and therefore to reduce the size of the game. However, removing more vertices increases the probability of obtaining macro actions prescribing long sequences of moves. This can worsen the performance since fixing the patroller's actions for long time intervals could introduce disadvantages.

Let us consider the class of utility loss abstractions. For a given λ we can find an abstraction as the result of the following linear program:

$$\max \sum_{i \in V} x_i \quad \text{s.t. constraints (1), (2), (3), (7), (8), (9), (10), (11), (12)}$$

$$s_{it} \leq (1-\lambda)dist(i,t) + \lambda d(t) \qquad \forall i \in \Lambda \setminus \Lambda_{\text{LLA}}, t \in T \tag{14}$$

$$s_{it} \leq (1-\lambda)dist(i,k) + \lambda d(t) \qquad \forall i \in \Lambda_{\text{LLA}}, t \in T, k \in D(i,t) \tag{15}$$

Constraints (14) apply the same principle of constraints (13) to vertices that are exclusively removable in lossy abstractions; constraints (15) apply to all vertices that are removable also in a lossless abstraction. For these vertices the minimum value of displacements accounts for the distance $dist(i,k)$ where $k \in D(i,t)$. Since we are considering lossy abstractions, the maximum displacement for each vertex is equal to the penetration time of the associated target. With respect to the game size, the parameter λ has the same meaning described for lossless abstractions. However, the removal of vertices introduces information loss increasing the probability for the intruder of having new candidate best responses with respect to the original game.

To evaluate how choosing an abstraction class and a value for λ in that class, we consider 100 patrolling settings (i.e., instances of G) with a size of approximately 50 vertices and a number of randomly located targets varying from 5% to 30% of the total number of vertices. For every setting we compute lossless and lossy abstractions for every value of λ with a resolution of 0.1 (by using AMPL and CPLEX), we apply the found abstraction to the original game and we compute the corresponding equilibrium (by using SNOPT) with a UNIX computer with dual quad–core 2.33GHz CPU and 8GB RAM. In Fig. 2 we report the averages of the abstraction values and of the computational time needed to find the equilibria (included the time for computing the abstractions that demonstrated to be negligible in all our experiments) as λ vary. Since values depend on the specific payoffs chosen in each patrolling setting, we normalize them in order to obtain consistent averages. Under the curves of abstraction values we report the percentages of terminations by a threshold of 30 m.

Initially, we observe that experimental results confirm the intuition behind the heuristics. Let us focus on utility lossless abstractions. As we expected, lossless abstractions always provide a better utility than the original game. The maximum value is at $\lambda = 0.9$ and the improvement w.r.t.

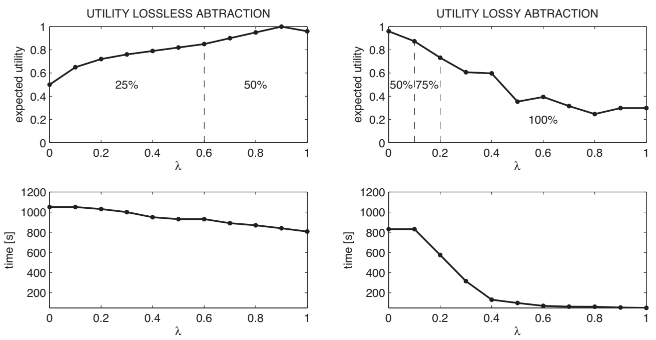

Figure 2: Abstractions values and resolution times.

$\nu(\varnothing)$ is about 100%. Therefore, utility lossless abstractions allow one to drastically improve the effectiveness of the patroller's strategies. The computational time strictly reduces with increasing value of λ, saving about 25% of the computational time for the maximal abstraction. Notice that by using abstractions with $\lambda \geq 0.6$ we can solve many game instances (25%) that would be not solvable without abstractions. Let us focus on utility loss abstractions. They can provide a utility both worse than $\nu(\varnothing)$ (on average for $\lambda \geq 0.5$, when the abstraction is very coarse) and better (on average for $\lambda \leq 0.4$, when the abstraction is close to the lossless one). The maximum value is at $\lambda = 0$ and the improvement w.r.t. to $\nu(\varnothing)$ is about 90%. The computational time strictly reduces with increasing values of λ, saving about 95% of the computational time needed for the maximal abstraction. Furthermore, by using abstractions with $\lambda \geq 0.2$ we can solve all the game instances. As the settings became larger (with more than 50 vertices) the termination probability reduces and only abstractions with λ close to 1 can be used (further experiments showed that with these abstractions the limit is at about 166 vertices and 5% of targets, usually due to excessive memory occupation). Therefore, utility loss abstractions allow one to solve much larger game instances than those solvable without abstractions.

Anytime Search in the Abstractions Space

The fact that with realistic settings the termination percentage is very low except for values of λ close to one with lossy abstractions pushes one to use $\lambda = 1$. Anyway, this choice could prevent one to find better solutions computable by a given deadline. Here, we pose the problem to search for the abstraction with the maximum value among those solvable in anytime fashion. The problem can be formulated as a variation of the optimization of a function in a continuous variable (i.e., λ) whose values are evaluated by simulation. No information on the function to optimize is available except for the heuristic information derived from the previous experimental results. To address this problem, we develop an algorithm inspired by (Vorobeychik and Wellman 2008) and based on adaptive grids and local search.

We define $\lambda' \in [0,2]$ such that , if $\lambda' < 1$, then lossless abstractions are used with $\lambda = \lambda'$, otherwise lossy abstractions are used with $\lambda = \lambda' - 1$. We denote by $X(\lambda')$ the abstraction produced given λ'. In our algorithm, we have a tuple ϕ that contains the samples $\lambda'_i \in [0,2]$ that have been evaluated and the corresponding value $\nu(X(\lambda'_i))$; it is defined as $\phi = \langle(\lambda'_1, \nu(X(\lambda'))), \ldots, (\lambda'_n, \nu(X(\lambda')))\rangle$. Elements in ϕ are in increasing order in λ'_i. Given a sample λ' we denote by λ'_+ the smallest sample larger than λ' among those in ϕ and by λ'_- the largest sample smaller than λ' among those in ψ. We have a tuple ψ that contains samples λ'_i that have not been evaluated yet. These sample are assigned an heuristic value $h(\lambda'_i)$ and are ordered in ψ according to it. We have finally two parameters: a temporal deadline ρ over the resolution of each single abstraction and a temperature $\tau \in [0, 1]$ whose use is explained below.

In the initialization, our algorithm starts by evaluating $\nu(\cdot)$ at $\lambda' = 2$ without imposing any temporal deadline and then it adds $(2, \nu(X(2)))$ to ϕ and $(0, h(0))$ where $h(0) = \nu(X(0))$. Call Δ the time spent to evaluate $\nu(X(2))$. From here on, the algorithm runs until there is remaining time and during its execution it stores the best abstraction found so far. With probability $1 - \tau$, the first element in ψ, say $\overline{\lambda}'$, is removed and it is evaluated imposing a temporal deadline ρ. To have a fine experimental evaluation, we normalize ρ w.r.t. Δ as $\frac{\rho}{\Delta}$. From here on, we consider ρ as a normalized parameter. With probability τ a random element of ψ is chosen. If the resolution terminates, then the element is introduced in ϕ after that the heuristic value $h(\overline{\lambda}')$ has been replaced by the actual value $\nu(X(\overline{\lambda}'))$. In addition, two new elements are added to ψ. The first (at right) is composed by the sample $\overline{\lambda}' + \frac{\overline{\lambda}'_+ - \overline{\lambda}'}{2}$ and h is $\frac{\nu(X(\overline{\lambda}')) + \nu(X(\overline{\lambda}'_+))}{2}$. The second (at left) is composed by the sample $\overline{\lambda}' - \frac{\overline{\lambda}' - \overline{\lambda}'_-}{2}$ and h is $\frac{\nu(X(\overline{\lambda}')) + \nu(X(\overline{\lambda}'_-))}{2}$. Essentially, these two samples refine the grain of the grid. If the evaluation of $\overline{\lambda}'$ exceeds ρ, $\overline{\lambda}'$ is added to ϕ with a value equal to that $\overline{\lambda}'_+$ and only the right sample is added to ψ. With opportune values of τ and ρ the algorithm is granted to converge to the optimal solution, being essentially a simulated annealing algorithm.

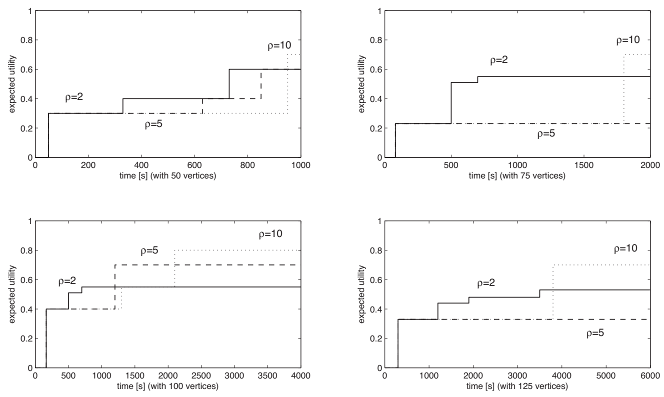

Figure 3: Anytime properties with different ρ at $\tau = 0.1$.

Fig. 3 reports the values of the best found abstractions in function of time for different values of normalized ρ, different sizes of settings (50, 75, 100, 125 vertices) and with $\tau = 0.1$ (when $\tau \leq 0.05$ and $\tau \geq 0.2$ the performance worsens). Call D the available time. As we expected, the larger the value of D/Δ the larger the value of the optimal ρ. From our experimental results, the best parametrization of the normalized ρ results $\rho \simeq \frac{D}{2.5\Delta}$.

Conclusions

In this paper, we studied the application of game theoretical abstractions to Patrolling Security Games (PSG). We defined significant classes of abstractions, discussing and experimentally evaluating their impact on the resolution of PSGs. We posed the problem of finding good abstractions and we proposed an automated method, based on an anytime search algorithm, to achieve this task. From our analysis, abstractions turned out to be an effective technique for simplifying the resolution of a PSG.

Future works include the study of new heuristics for addressing the limitations associated to the resolution of a PSG and an extension of this approach to the more general class of security games. In addition, more sophisticated techniques can be used for the anytime algorithm, such as using a time variable deadline on the resolution of the single abstraction.

References

Basilico, N.; Gatti, N.; Rossi, T.; Ceppi, S.; and Amigoni, F. 2009. Extending algorithms for mobile robot patrolling in the presence of adversaries to more realistic settings. In *IAT*, 557–564.

Basilico, N.; Gatti, N.; and Amigoni, F. 2009. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*, 57–64.

Conitzer, V., and Sandholm, T. 2006. Computing the optimal strategy to commit to. In *EC*, 82–90.

Gilpin, A., and Sandholm, T. 2007. Lossless abstraction of imperfect information games. *J ACM* 54(5).

Gilpin, A.; Sandholm, T.; and Sørensen, T. 2008. A heads-up no-limit texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *AAMAS*, 911–918.

Paruchuri, P.; Pearce, J.; Tambe, M.; Ordonez, F.; and Kraus, S. 2007. An efficient heuristic approach for security against multiple adversaries. In *AAMAS*, 311–318.

Paruchuri, P.; Pearce, J.; Marecki, J.; Tambe, M.; Ordonez, F.; and Kraus, S. 2008. Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games. In *AAMAS*, 895–902.

Pita, J.; Jain, M.; Marecki, J.; Ordonez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In *AAMAS*, 125–132.

Tsai, J.; Yin, Z.; Kwak, J.; Kempe, D.; Kiekintveld, C.; and Tambe, M. 2010. Urban security: Game-theoretic resource allocation in networked physical domains. In *AAAI*, 881 – 886.

Vorobeychik, Y., and Wellman, M. 2008. Stochastic search methods for nash equilibrium approximation in simulation-based games. In *AAMAS*, 1055–1062.